**Taller práctico.**

**Actividad N 25.**

**MVC\_CRUD parte 3.**

**Objetivo de la actividad:** Codificar las interfaces gráficas y módulos usando PHP y el servidor apache aplicando el concepto de MVC\_CRUD

**Paso 12.**

Continuando con el ejercicio, se especifican los controladores del proyecto, este código debe ubicarse en la carpeta Controller del proyecto. Creamos proveedor.controller.php el nombre debe guardarse de esta forma.

|  |
| --- |
| proveedor.controller.php |
| <?php  //Se incluye el modelo donde conectará el controlador.  require\_once "model/proveedor.php";  class ProveedorController{  private $model;  //Creación del modelo  public function \_\_CONSTRUCT(){  $this->model = new proveedor();  }  //Llamado plantilla principal  public function Index(){  require\_once 'view/header.php';  require\_once 'view/proveedor/proveedor.php';  require\_once 'view/footer.php';  }  //Llamado a la vista proveedor-editar  public function Crud( ){  $pvd = new proveedor();  //Se obtienen los datos del proveedor a editar.  if(isset($\_REQUEST['nit'])){  $pvd = $this->model->Obtener($\_REQUEST['nit']);  }  //Llamado de las vistas.  require\_once 'view/header.php';  require\_once 'view/proveedor/proveedor-editar.php';  require\_once 'view/footer.php';  } |

**Paso 13.**

Ahora se construye el llamado a la vista proveedor-nuevo

|  |
| --- |
| proveedor.controller.php (continuación código paso 12) |
| //Llamado a la vista proveedor-nuevo  public function Nuevo(){  $pvd = new proveedor();  //Llamado de las vistas.  require\_once 'view/header.php';  require\_once 'view/proveedor/proveedor-nuevo.php';  require\_once 'view/footer.php';  } |

**Paso 14.**

Este código complementa la acción solicitada anteriormente, es un método que registrar al modelo un nuevo proveedor.

|  |
| --- |
| proveedor.controller.php (continuación código paso 13) |
| //Método que registrar al modelo un nuevo proveedor.  public function Guardar(){  $pvd = new proveedor();  //Captura de los datos del formulario (vista)  $pvd->nit = $\_REQUEST['nit'];  $pvd->razonS = $\_REQUEST['razonS'];  $pvd->dir = $\_REQUEST['dir'];  $pvd->tel = $\_REQUEST['tel'];  //Registro al modelo proveedor.  $this->model->Registrar($pvd);  //header() es usado para enviar encabezados HTTP sin formato.  //”Location:” No solamente envía el encabezado al navegador, sino que  //también devuelve el código de status (302) REDIRECT al  //navegador  header('Location: index.php');  } |

**Paso 15.**

Este código complementa la acción solicitada anteriormente, es un método que modifica el modelo de un proveedor.

|  |
| --- |
| proveedor.controller.php (continuación código paso 14) |
| //Método que modifica el modelo de un proveedor.  public function Editar(){  $pvd = new proveedor();  $pvd->nit = $\_REQUEST['nit'];  $pvd->razonS = $\_REQUEST['razonS'];  $pvd->dir = $\_REQUEST['dir'];  $pvd->tel = $\_REQUEST['tel'];  $this->model->Actualizar($pvd);  header('Location: index.php');  } |

**Paso 16.**

Este código complementa la acción solicitada anteriormente, es un método que elimina la tupla proveedor con el NIT dado.

|  |
| --- |
| proveedor.controller.php (continuación código paso 15) |
| //Método que elimina la tupla proveedor con el NIT dado.  public function Eliminar(){  $this->model->Eliminar($\_REQUEST[‘nit’]);  header(‘Location: index.php’);  }  }//no borrar ya que es la llave que cierra el class. |

Como actividad debes:

1. Crear los elementos descritos desde el paso 6, hasta el paso 11. Has captura de pantalla según la siguiente tabla: (recuerda hacer la descripción de cada paso empleado una buena redacción)

|  |
| --- |
| Captura de pantalla paso12 |
| Ahora dentro de la carpeta controller creamos un archivo que será “proveedor.controller.php” es un controlador tendrá una clase llamada ProveedorController. Este controlador se utiliza para interactuar con la tabla proveedor en de la base de datos.  El controlador ProveedorController tiene tres métodos:  CONSTRUCT(): Este método es el constructor de la clase y se utiliza para crear un objeto de la clase proveedor.  Index(): Este método se utiliza para cargar la plantilla principal de la aplicación.  Crud(): Este método se utiliza para cargar la vista proveedor-editar.php y obtener los datos del proveedor a editar.  En el método Crud(), se crea un objeto de la clase proveedor y se obtienen los datos del proveedor a editar utilizando el método Obtener() de la clase proveedor. Luego, se cargan las vistas necesarias para mostrar los datos del proveedor en un formulario. |
| Captura de pantalla paso13 |
| Creamos otro método llamado Nuevo() que pertenece a la clase ProveedorController en el lenguaje de programación PHP. Este método se utiliza para cargar la vista proveedor-nuevo.php, que se utiliza para agregar un nuevo proveedor a la tabla proveedor en una base de datos.  En el método Nuevo(), se crea un objeto de la clase proveedor. Luego, se cargan las vistas necesarias para mostrar un formulario que permita al usuario ingresar los datos del nuevo proveedor. |
| Captura de pantalla paso14 |
| un método llamado Guardar(). Este método se utiliza para guardar los datos de un proveedor en la base de datos.  En el método Guardar(), se crea un objeto de la clase proveedor y se capturan los datos del formulario (vista) utilizando la variable global $\_REQUEST. Luego, se asignan los valores capturados a los atributos del objeto proveedor. A continuación, se registra el proveedor en la base de datos utilizando el método Registrar() del modelo proveedor. Finalmente, se redirige al usuario a la página index.php utilizando la función header(). |
| Captura de pantalla paso15 |
| un método llamado Editar(). Se utiliza para actualizar los datos de un proveedor en la tabla proveedor de una base de datos.  En el método Editar(), se crea un objeto de la clase proveedor y se capturan los datos del formulario (vista) utilizando la variable global $\_REQUEST. Luego, se asignan los valores capturados a los atributos del objeto proveedor. A continuación, se actualizan los datos del proveedor en la tabla proveedor utilizando el método Actualizar() del modelo proveedor. Finalmente, se redirige al usuario a la página index.php utilizando la función header(). |
| Captura de pantalla paso16 |
| Por ultimo un método llamado Eliminar(). Este método se utiliza para eliminar un registro de la tabla proveedor en una base de datos.  En el método Eliminar(), se llama al método Eliminar() del modelo proveedor y se pasa el valor del parámetro nit obtenido de la variable global $\_REQUEST. Luego, se redirige al usuario a la página index.php utilizando la función header(). |

1. ¿para que se usa class ProveedorController? Explica en un párrafo de 57 palabras.

La clase "ProveedorController" se utiliza para controlar las acciones relacionadas con la gestión de proveedores. Este controlador se encarga de manejar las solicitudes y las vistas relacionadas con los proveedores, como listar proveedores, editar proveedores, agregar nuevos proveedores, guardar cambios en la base de datos, y eliminar proveedores.

1. ¿para que se usa header()? Explica en un párrafo de 57 palabras.

La función header() en PHP se utiliza para enviar encabezados HTTP sin formato. Los encabezados HTTP son información adicional que se envía junto con una solicitud o respuesta HTTP. La función header() se utiliza para enviar encabezados personalizados, como la redirección a otra página, la especificación del tipo de contenido, la codificación de caracteres y más. Es importante tener en cuenta que la función header() debe ser llamada antes de mostrar cualquier cosa por pantalla, etiquetas HTML o líneas en blanco desde un fichero o desde PHP. La función header() también se utiliza para enviar códigos de estado HTTP al navegador.

1. ¿Qué es el composer? Explica en un párrafo de 500 palabras según sea el caso.

Composer es un administrador de dependencias para el lenguaje de programación PHP. Es una herramienta que se utiliza para administrar y descargar bibliotecas y paquetes de código fuente de terceros en un proyecto PHP. Composer se basa en el concepto de paquete, que es una biblioteca o conjunto de archivos que proporcionan una funcionalidad específica. Los paquetes se pueden instalar y actualizar automáticamente utilizando Composer, lo que facilita la gestión de dependencias en proyectos PHP.

Composer utiliza un archivo llamado composer.json para definir las dependencias del proyecto. Este archivo contiene información sobre los paquetes requeridos por el proyecto, así como información adicional, como la versión de PHP requerida y los repositorios donde se pueden encontrar los paquetes.

Una de las ventajas clave de Composer es su capacidad para manejar automáticamente las dependencias del proyecto. Cuando se instala un paquete, Composer también instala todas las dependencias requeridas por ese paquete1. Además, Composer permite a los desarrolladores especificar versiones específicas de paquetes y actualizar fácilmente a nuevas versiones cuando están disponibles.

Otra ventaja importante de Composer es su amplia gama de paquetes disponibles. Hay miles de paquetes disponibles en el repositorio oficial de paquetes de Composer, así como en otros repositorios públicos y privados3. Esto permite a los desarrolladores agregar fácilmente nuevas funcionalidades a sus proyectos sin tener que escribir todo el código desde cero.